DupHunter: Flexible High-Performance Deduplication for Docker Registries

Nannan Zhao, Hadeel Albahar, Subil Abraham, Keren Chen, Vasily Tarasov, Dimitrios Skourtis, Lukas Rupprecht, Ali Anwar, and Ali R. Butt
Containers are ubiquitous

- **OS**: Ubuntu, Alpine, etc.
- **Database**: Oracle, MySQL, PostgreSQL
- **Web server**: Nginx
- **Cache**: Redis
- **Serverless**: AWS Lambda, Azure Functions, Google Cloud Functions
- **Deep learning**: TensorFlow, PyTorch, Caffe
- **Big data**: Spark, Hadoop, Storm
- **Languages**: Ruby, Python, Go, Java
Application containerization is becoming a significant market player.

Application Containers: Total Market Revenue ($M)

<table>
<thead>
<tr>
<th>Year</th>
<th>Revenue ($M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017</td>
<td>$1,124</td>
</tr>
<tr>
<td>2018</td>
<td>$1,567</td>
</tr>
<tr>
<td>2019</td>
<td>$2,126</td>
</tr>
<tr>
<td>2020</td>
<td>$2,755</td>
</tr>
<tr>
<td>2021</td>
<td>$3,467</td>
</tr>
<tr>
<td>2022</td>
<td>$4,311</td>
</tr>
</tbody>
</table>

30.8% 2017-22 CAGR

Source: 451 Research’s Market Monitor: Cloud-Enabling Technologies - Application Containers, November 2018
Oracle Database Enterprise Edition

By Oracle • Updated 3 years ago

Oracle Database 12c Enterprise Edition

Container Docker Certified Linux x86-64 Databases

MySQL Server Enterprise Edition

By Oracle • Updated 2 years ago

The world's most popular open source database system

Container Docker Certified Linux x86-64 Databases

couchbase

Updated 31 minutes ago

Couchbase Server is a NoSQL document database with a distributed architecture.

Container Linux x86-64 Storage Application Frameworks

postgres

Updated 33 minutes ago

Postgres

Nannan Zhao znannan1@vt.edu
Docker image dataset is growing fast!
How to efficiently manage the ever-growing image dataset for Docker registries?
Our contribution: DupHunter—a framework to deduplicate images in Docker registries

- We make two key observations:
  1. Container images exhibit a lot of redundancy.
  2. User access pattern is predictable.

- We design DupHunter to work with compressed images and provide layer deduplication and reduce layer restore overhead.

- We evaluate DupHunter with representative real world workloads. Compared to the state of the art, DupHunter:
  - reduces storage space by up to 6.9x.
  - reduces the GET layer latency up to 2.8x.
Overview of Docker

- Docker container is a self-contained executable package, that is:
  - Lightweight
  - Portable
  - Provides Isolation

- Docker registry:
  - Stores Docker images
  - Supports fast distribution
  - Facilitates easy deployment
Key observation I: Image dataset has large amount of redundant files

- Container images have a lot of redundancy.
  - 97% of files across layers are duplicates!

- Existing technologies such as Jdupes, VDO, Btrfs, ZFS, and Ceph are unable to harness this redundancy.

Does not help!
Key observation 1: Image dataset has large amount of redundant files

- Container images have a lot of redundancy.
  - 97% of files across layers are duplicates!
- Existing technologies such as Jdupes, VDO, Btrfs, ZFS, and Ceph are unable to harness this redundancy.

Compressed layer dataset

Does not help!
Key observation I: Image dataset has large amount of redundant files

- Container images have a lot of redundancy.
  - 97% of files across layers are duplicates!

- Existing technologies such as Jdupes, VDO, Btrfs, ZFS, and Ceph are unable to harness this redundancy.

Does not help!
Key observation I: Image dataset has large amount of redundant files

- Container images have a lot of redundancy.
  - 97% of files across layers are duplicates!

- Existing technologies such as Jdupes, VDO, Btrfs, ZFS, and Ceph are unable to harness this redundancy.

Diagram:
- Compressed layer dataset → Unpack → Uncompressed layer dataset → Decompress
Key observation I: Image dataset has large amount of redundant files

- Container images have a lot of redundancy.
  - 97% of files across layers are duplicates!
- Existing technologies such as Jdupes, VDO, Btrfs, ZFS, and Ceph are unable to harness this redundancy.

Unpack → Decompress → Deduplicate

Compressed layer dataset

Uncompressed layer dataset

Does not help!
Key observation I: Image dataset has large amount of redundant files

- Container images have a lot of redundancy.
  - 97% of files across layers are duplicates!

- Existing technologies such as Jdupes, VDO, Btrfs, ZFS, and Ceph are unable to harness this redundancy.

Does not help!

Compressed layer dataset

Unpack

Decompress

Uncompressed layer dataset

Deduplicate
Key observation I: Image dataset has large amount of redundant files

- Container images have a lot of redundancy.
  - 97% of files across layers are duplicates!

- Existing technologies such as Jdupes, VDO, Btrfs, ZFS, and Ceph are unable to harness this redundancy.

Does not help! Compressed layer dataset
Decompress Unpack Deduplicate
Uncompressed layer dataset
Reduces space by up to 4X

Nannan Zhao  znannan1@vt.edu
Key observation I: Image dataset has large amount of redundant files

- Container images have a lot of redundancy.
  - 97% of files across layers are duplicates!
- Existing technologies such as Jdupes, VDO, Btrfs, ZFS, and Ceph are unable to harness this redundancy.

Layer restore incurs considerable overhead for layer pulling latency up to 98x!
Key observation II: Predictable user access pattern

- We observe a consistent user pulling pattern: Pull manifest first, then layers, but not all of the layers will be pulled.
- We performed a quantitative study using a 75-day IBM Cloud Registry workload with 7 availability zones.
Key observation II: Predictable user access pattern

- We observe a consistent user pulling pattern: Pull manifest first, then layers, but not all of the layers will be pulled.
- We performed a quantitative study using a 75-day IBM Cloud Registry workload with 7 availability zones.
Key observation II: Predictable user access pattern

- We observe a consistent user pulling pattern: Pull manifest first, then layers, but not all of the layers will be pulled.
- We performed a quantitative study using a 75-day IBM Cloud Registry workload with 7 availability zones.

![Diagram showing layers ratio by GET Layer count for different regions (Dal, Dev, Fra, Lon, Sta, Syd). The majority of layers are only fetched once by the same client.](image)

Majority of layers are only fetched once by the same client.

Nannan Zhao, zha363@vt.edu
Key observation II-b: User repulling pattern can also be predicted

![Graph showing repulling probability vs clients ratio for different regions.](image)

Nannan Zhao  znannan1@vt.edu
Key observation II-b: User repulling pattern can also be predicted

Half of the clients have a repull probability less than 0.2 → many clients pull a layer only once.
Key observation II-b: User repulling pattern can also be predicted.
Key observation II-b: User repulling pattern can also be predicted.
Key observation II-b: User repulling pattern can also be predicted

User repulling pattern is either pull-once or always-pull → we can predict which layers to pull.
Key observation II-c: Layer preconstruction is possible
Key observation II-c: Layer preconstruction is possible

Layer preconstruction can significantly reduce layer restore overhead.

Nannan Zhao  znannan1@vt.edu
DupHunter architecture

- Distributed metadata database
- Server A
- Server B
- Server C
- Server D
- Storage cluster
- Registry REST API
- Local storage system

Clients

Nannan Zhao  znannan1@vt.edu
Reducing overhead in DupHunter

1. Support multiple replica deduplication modes.
2. Facilitate parallel layer reconstruction.
3. Enable proactive layer prefetching/preconstruction.
DupHunter supports multiple replica deduplication modes

- **B-mode $n$:** Basic deduplication mode $n$
  - Keep $n$ layer replicas intact.
  - Deduplicate the remaining $R-n$ layer replicas ($R =$ layer replication level).

- **S-mode:** Selective deduplication mode
  - The number of intact layer replicas proportional to the layer’s popularity.
  - Hot layers have more intact replicas.

Nannan Zhao  znannan1@vt.edu
DupHunter supports multiple replica deduplication modes

- **B-mode $n$:** Basic deduplication mode $n$
  - Keep $n$ layer replicas intact.
  - Deduplicate the remaining $R-n$ layer replicas ($R$ = layer replication level).

- **S-mode:** Selective deduplication mode
  - The number of intact layer replicas proportional to the layer’s popularity.
  - Hot layers have more intact replicas.
DupHunter facilitates parallel layer reconstruction

- **Slice**: Set of all the files on a server belonging to a layer.
  - Distributed evenly across the cluster.
  - Speed up layer reconstruction via parallel processing of slices.
DupHunter enables prefetching/preconstruction of layers

- **Prefetch cache** to prefetch layers and hide disk I/Os.
- **Preconstruct cache** to store preconstruct layers and hide layer restore overhead.
# Deduplicating layers

<table>
<thead>
<tr>
<th>Header</th>
<th>Content fingerprint</th>
</tr>
</thead>
<tbody>
<tr>
<td>h1</td>
<td>f1</td>
</tr>
<tr>
<td>h2</td>
<td>f2</td>
</tr>
<tr>
<td>h3</td>
<td>f3</td>
</tr>
<tr>
<td>h4</td>
<td>f4</td>
</tr>
<tr>
<td>h5</td>
<td>f5</td>
</tr>
<tr>
<td>h6</td>
<td>f6</td>
</tr>
</tbody>
</table>

Layer tar archive \( L1 \)

File entries
Deduplicating layers

Layer tar archive $L_1$

Header

<table>
<thead>
<tr>
<th>Header</th>
<th>Content fingerprint</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td>$f_1$</td>
</tr>
<tr>
<td>$h_2$</td>
<td>$f_2$</td>
</tr>
<tr>
<td>$h_3$</td>
<td>$f_3$</td>
</tr>
<tr>
<td>$h_4$</td>
<td>$f_4$</td>
</tr>
<tr>
<td>$h_5$</td>
<td>$f_5$</td>
</tr>
<tr>
<td>$h_6$</td>
<td>$f_6$</td>
</tr>
</tbody>
</table>

File entries

File index

<table>
<thead>
<tr>
<th>Id</th>
<th>$r_1$</th>
<th>$r_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>A:/../..</td>
<td>B:/../..</td>
</tr>
<tr>
<td>$f_2$</td>
<td>B:/../..</td>
<td>C:/../..</td>
</tr>
</tbody>
</table>
Deduplicating layers

Layer tar archive L1

File entries

Header

Content fingerprint

Duplicate / Shared files

Unique files

Layer tar archive L1

File index

<table>
<thead>
<tr>
<th>Id</th>
<th>r1</th>
<th>r2</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>A: ../..</td>
<td>B: ../..</td>
</tr>
<tr>
<td>f2</td>
<td>B: ../..</td>
<td>C: ../..</td>
</tr>
</tbody>
</table>
Deduplicating layers

Layer tar archive

File entries

Header

Content fingerprint

File index

Id | r1   | r2   |
---|------|------|
f1 | A:/../ | B:/../ |
f2 | B:/../ | C:/../ |

Stored file replicas

D-server A

D-server B

D-server C

Duplicate / Shared files

Unique files

f1' f1 f2 f3' f1' f2' f3

Nannan Zhao  znannan1@vt.edu
Deduplicating layers

Layer tar archive $L_1$

File entries

<table>
<thead>
<tr>
<th>Header</th>
<th>Content fingerprint</th>
</tr>
</thead>
<tbody>
<tr>
<td>h1</td>
<td>f1</td>
</tr>
<tr>
<td>h2</td>
<td>f2</td>
</tr>
<tr>
<td>h3</td>
<td>f3</td>
</tr>
<tr>
<td>h4</td>
<td>f4</td>
</tr>
<tr>
<td>h5</td>
<td>f5</td>
</tr>
<tr>
<td>h6</td>
<td>f6</td>
</tr>
</tbody>
</table>

Duplicate / Shared files

<table>
<thead>
<tr>
<th>D1</th>
<th>D2</th>
<th>D3</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>f2</td>
<td>f3</td>
</tr>
<tr>
<td>f4</td>
<td>f5</td>
<td>f6</td>
</tr>
<tr>
<td>f6'</td>
<td>f4'</td>
<td>f5'</td>
</tr>
</tbody>
</table>

Unique files

<table>
<thead>
<tr>
<th>Stored file replicas</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1'</td>
</tr>
<tr>
<td>f2'</td>
</tr>
<tr>
<td>f3'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Newly added file replicas</th>
</tr>
</thead>
<tbody>
<tr>
<td>f4</td>
</tr>
<tr>
<td>f5</td>
</tr>
</tbody>
</table>

File index

<table>
<thead>
<tr>
<th>Id</th>
<th>r1</th>
<th>r2</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>A:/..</td>
<td>B:/..</td>
</tr>
<tr>
<td>f2</td>
<td>B:/..</td>
<td>C:/..</td>
</tr>
</tbody>
</table>

Nannan Zhao  znannan1@vt.edu
Deduplicating layers

Layer tar archive

<table>
<thead>
<tr>
<th>h1</th>
<th>f1</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2</td>
<td>f2</td>
</tr>
<tr>
<td>h3</td>
<td>f3</td>
</tr>
<tr>
<td>h4</td>
<td>f4</td>
</tr>
<tr>
<td>h5</td>
<td>f5</td>
</tr>
<tr>
<td>h6</td>
<td>f6</td>
</tr>
</tbody>
</table>

Layer tar archive

Header

Content fingerprint

Duplicate / Shared files

Unique files

Stored file replicas

Newly added file replicas

D-server A

D-server B

D-server C

File entries

File index

<table>
<thead>
<tr>
<th>Id</th>
<th>r1</th>
<th>r2</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>A:/../.. B:/../..</td>
<td></td>
</tr>
<tr>
<td>f2</td>
<td>B:/../.. C:/../..</td>
<td></td>
</tr>
</tbody>
</table>
Deduplicating layers

Layer tar archive \textit{L1}

File entries

- \textit{h1}: \textit{f1}
- \textit{h2}: \textit{f2}
- \textit{h3}: \textit{f3}
- \textit{h4}: \textit{f4}
- \textit{h5}: \textit{f5}
- \textit{h6}: \textit{f6}

Header

Content fingerprint

Duplicate / Shared files

Unique files

- \textit{f1}
- \textit{f2}
- \textit{f3}
- \textit{f4}
- \textit{f5}
- \textit{f6}

Stored file replicas

Newly added file replicas

D-server A

D-server B

D-server C

File index

<table>
<thead>
<tr>
<th>Id</th>
<th>r1</th>
<th>r2</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{f1}</td>
<td>A:/../..</td>
<td>B:/../..</td>
</tr>
<tr>
<td>\textit{f2}</td>
<td>B:/../..</td>
<td>C:/../..</td>
</tr>
</tbody>
</table>

Slice recipe

Id: \textit{L1}:A:P

Header

Content pointer

- \textit{h2}: \textit{f2}
- \textit{h5}: \textit{f5}

Nannan Zhao \ znannan1@vt.edu
Deduplicating layers

Layer tar archive \( L1 \)

**File entries**

<table>
<thead>
<tr>
<th>Header</th>
<th>Content fingerprint</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h1 )</td>
<td>( f1 )</td>
</tr>
<tr>
<td>( h2 )</td>
<td>( f2 )</td>
</tr>
<tr>
<td>( h3 )</td>
<td>( f3 )</td>
</tr>
<tr>
<td>( h4 )</td>
<td>( f4 )</td>
</tr>
<tr>
<td>( h5 )</td>
<td>( f5 )</td>
</tr>
<tr>
<td>( h6 )</td>
<td>( f6 )</td>
</tr>
</tbody>
</table>

**Duplicate / Shared files**

- \( f1 \)
- \( f2 \)
- \( f3 \)

**Unique files**

- \( f4 \)
- \( f5 \)
- \( f6 \)

**Stored file replicas**

- \( f1' \)
- \( f2' \)
- \( f3' \)

**Newly added file replicas**

- \( f4' \)
- \( f5' \)

**Layer recipe**

Id: \( L1 \)
Master: A
Workers: [A, B, C]

**Slice recipe**

Id: \( L1 : : A : : P \)

<table>
<thead>
<tr>
<th>Header</th>
<th>Content pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h2 )</td>
<td>( f2 )</td>
</tr>
<tr>
<td>( h5 )</td>
<td>( f5 )</td>
</tr>
</tbody>
</table>

File index

<table>
<thead>
<tr>
<th>Id</th>
<th>( r1 )</th>
<th>( r2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f1 )</td>
<td>A:/../..</td>
<td>B:/../..</td>
</tr>
<tr>
<td>( f2 )</td>
<td>B:/../..</td>
<td>C:/../..</td>
</tr>
</tbody>
</table>

Nannan Zhao  znannan1@vt.edu
Restoring layers

Slice constructor

<table>
<thead>
<tr>
<th>A</th>
<th>File I/O stream</th>
<th>archive</th>
<th>compress</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Tar stream</td>
<td>archive</td>
<td>compress</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>archive</td>
<td>compress</td>
</tr>
</tbody>
</table>

Layer constructor

concatenate

Nannan Zhao  znannan1@vt.edu
Caching and preconstricting layers

- ILmap: Maps image to its containing layer set.
- ULmap: Maps user to the layers that the user has accessed and the corresponding pull count.
Caching and preconstructing layers

- **ILmap**: Maps image to its containing layer set.
- **ULmap**: Maps user to the layers that the user has accessed and the corresponding pull count.
Caching and preconstructing layers

- ILmap: Maps image to its containing layer set.
- ULmap: Maps user to the layers that the user has accessed and the corresponding pull count.
Caching and preconstructing layers

- **ILmap**: Maps image to its containing layer set.
- **ULmap**: Maps user to the layers that the user has accessed and the corresponding pull count.
Caching and preconstructing layers

- **ILmap**: Maps image to its containing layer set.
- **ULmap**: Maps user to the layers that the user has accessed and the corresponding pull count.

![Diagram of image and user sets]

Will pull
Caching and preconstructing layers

- ILmap: Maps image to its containing layer set.
- ULmap: Maps user to the layers that the user has accessed and the corresponding pull count.

\[ S_\Delta = ILmap[r.img] - ULmap[r.addr] \]
Caching and preconstructing layers

- **ILmap**: Maps image to its containing layer set.
- **ULmap**: Maps user to the layers that the user has accessed and the corresponding pull count.

\[ S_\Delta = \text{ILmap}[r.\text{img}] - \text{ULmap}[r.\text{addr}] \]
Caching and preconstructing layers

- ILmap: Maps image to its containing layer set.
- ULmap: Maps user to the layers that the user has accessed and the corresponding pull count.

\[ S_\Delta = \text{ILmap}[r.img] - \text{ULmap}[r.addr] \]
Caching and preconstructing layers

- **ILmap**: Maps image to its containing layer set.
- **ULmap**: Maps user to the layers that the user has accessed and the corresponding pull count.

\[
S_\Delta = ILmap[r.img] - ULmap[r.addr]
\]
Caching and preconstrcucting layers

- **ILmap**: Maps image to its containing layer set.
- **ULmap**: Maps user to the layers that the user has accessed and the corresponding pull count.

\[ S_\Delta = ILmap[r.img] - ULmap[r.addr] \]
Caching and preconstructing layers

- ILmap: Maps image to its containing layer set.
- ULmap: Maps user to the layers that the user has accessed and the corresponding pull count.

\[
S_\Delta = ILmap[r.img] - ULmap[r.addr] \quad S_n = ILmap[r.img] \cap ULmap[r.addr]
\]
Cache handling in tiered storage

Tier 1
Primary cluster

P-server A

P-server B

Nannan Zhao  znannan1@vt.edu
Cache handling in tiered storage

Tier 1
Primary cluster

- P-server A
- P-server B

Tier 2
Deduplication cluster

- D-server C
- D-server D
Cache handling in tiered storage

**Tier 1**
Primary cluster

- Cache
- P-server A
- P-server B

**Tier 2**
Deduplication cluster

- D-server C
- D-server D

*L1 Prefetch cache*
Cache handling in tiered storage

**Tier 1**
Primary cluster
- Cache
- Layer store
- P-server A
- Cache
- Layer store
- P-server B
- L1 Prefetch cache
- L2 Layer store

**Tier 2**
Deduplication cluster
- D-server C
- D-server D

Nannan Zhao  znannan1@vt.edu
Cache handling in tiered storage

Tier 1
Primary cluster

- Cache
- Layer store
  - P-server A
- Cache
  - Layer store
    - P-server B

Tier 2
Deduplication cluster

- Stage area
  - D-server C
- Stage area
  - D-server D

L1 Prefetch cache
L2 Layer store
L3 Layer stage area
Cache handling in tiered storage

Tier 1
Primary cluster
- Cache
- Layer store
  - P-server A

Tier 2
Deduplication cluster
- Stage area
  - D-server C
- Stage area
  - D-server D

Layer store
- Layer store
  - P-server B

L1 Prefetch cache
L2 Layer store
L3 Layer stage area

Nannan Zhao  znannan1@vt.edu
Cache handling in tiered storage

**Tier 1**
Primary cluster
- Cache
- Layer store
  - P-server A

**Tier 2**
Deduplication cluster
- Stage area
  - Cache
  - D-server C
- Stage area
  - Cache
  - D-server D

**Cache layers**
- **L1** Prefetch cache
- **L2** Layer store
- **L3** Layer stage area
- **L4** Preconstruct cache
Cache handling in tiered storage

Tier 1
Primary cluster
- Cache
- Layer store
- P-server A

Tier 2
Deduplication cluster
- Stage area
- Cache
- File store
- D-server C

L1 Prefetch cache
L2 Layer store
L3 Layer stage area
L4 Preconstruct cache
L5 File store

Nannan Zhao znannan1@vt.edu
Evaluation

❑ Workloads used:
  ▪ Traces from IBM registries: Dal, Fra, Lon, and Syd availability zones
  ▪ Dataset from Docker Hub

❑ Schemes studied:
  ▪ **Baseline**: No deduplication
  ▪ **B-mode n**: $n (1-3)$ replicas are preserved; $3 - n$ deduplicated
  ▪ **S-mode**: intact layer replicas proportional to the layer’s popularity
  ▪ **B-mode 0**: deduplicate all layer replicas, under a given replication policy
    • **GF-R**: global file-level deduplication
    • **GF+LB-R**: global file-level deduplication and local block-level deduplication
    • **GB-EC**: global block-level deduplication under erasure coding
## Deduplication ratio vs. performance

<table>
<thead>
<tr>
<th>Mode</th>
<th>Dedup. ratio</th>
<th>Performance improvement (P-servers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-mode 1</td>
<td>1.5</td>
<td>1.6×</td>
</tr>
<tr>
<td>S-mode</td>
<td>1.3</td>
<td>2×</td>
</tr>
<tr>
<td>B-mode 2</td>
<td>1.2</td>
<td>2.6×</td>
</tr>
<tr>
<td>B-mode 3</td>
<td>1</td>
<td>2.8×</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B-mode 0</th>
<th>Dedup ratio</th>
<th>Performance degradation (D-servers)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GF-R</strong></td>
<td>2.1</td>
<td>-1.03 ×</td>
</tr>
<tr>
<td><strong>GF+LB-R</strong></td>
<td>3.0</td>
<td>-2.87 ×</td>
</tr>
<tr>
<td><strong>GB-EC</strong></td>
<td>6.9</td>
<td>-6.37 ×</td>
</tr>
</tbody>
</table>

Nannan Zhao  znannan1@vt.edu
## Deduplication ratio vs. performance

<table>
<thead>
<tr>
<th>Mode</th>
<th>Dedup. ratio</th>
<th>Performance improvement (P-servers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-mode 1</td>
<td>1.5</td>
<td>1.6×</td>
</tr>
<tr>
<td>S-mode</td>
<td>1.3</td>
<td>2×</td>
</tr>
<tr>
<td>B-mode 2</td>
<td>1.2</td>
<td>2.6×</td>
</tr>
<tr>
<td>B-mode 3</td>
<td>1</td>
<td>2.8×</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B-mode 0</th>
<th>Dedup ratio</th>
<th>Performance degradation (D-servers)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GF-R</strong> (Global file-level [3 replicas])</td>
<td>2.1</td>
<td>-1.03 ×</td>
</tr>
<tr>
<td><strong>GF+LB-R</strong> (Global file- and local block-level [3 replicas])</td>
<td>3.0</td>
<td>-2.87 ×</td>
</tr>
<tr>
<td><strong>GB-EC</strong> (Global block-level [Erasure coding])</td>
<td>6.9</td>
<td>-6.37 ×</td>
</tr>
</tbody>
</table>
# Deduplication ratio vs. performance

<table>
<thead>
<tr>
<th>Mode</th>
<th>Dedup. ratio</th>
<th>Performance improvement (P-servers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-mode 1</td>
<td>1.5</td>
<td>1.6×</td>
</tr>
<tr>
<td>S-mode</td>
<td>1.3</td>
<td>2×</td>
</tr>
<tr>
<td>B-mode 2</td>
<td>1.2</td>
<td>2.6×</td>
</tr>
<tr>
<td>B-mode 3</td>
<td>1</td>
<td>2.8×</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B-mode 0</th>
<th>Dedup ratio</th>
<th>Performance degradation (D-servers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GF-R (Global file-level [3 replicas])</td>
<td>2.1</td>
<td>-1.03 ×</td>
</tr>
<tr>
<td>GF+LB-R (Global file- and local block-level [3 replicas])</td>
<td>3.0</td>
<td>-2.87 ×</td>
</tr>
<tr>
<td>GB-EC (Global block-level [Erasure coding])</td>
<td>6.9</td>
<td>-6.37 ×</td>
</tr>
</tbody>
</table>
# Deduplication ratio vs. performance

<table>
<thead>
<tr>
<th>Mode</th>
<th>Dedup. ratio</th>
<th>Performance improvement (P-servers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-mode 1</td>
<td>1.5</td>
<td>1.6×</td>
</tr>
<tr>
<td>S-mode</td>
<td>1.3</td>
<td>2×</td>
</tr>
<tr>
<td>B-mode 2</td>
<td>1.2</td>
<td>2.6×</td>
</tr>
<tr>
<td>B-mode 3</td>
<td>1</td>
<td>2.8×</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B-mode 0</th>
<th>Dedup ratio</th>
<th>Performance degradation (D-servers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GF-R (Global file-level [3 replicas])</td>
<td>2.1</td>
<td>-1.03 ×</td>
</tr>
<tr>
<td>GF+LB-R (Global file- and local block-level [3 replicas])</td>
<td>3.0</td>
<td>-2.87×</td>
</tr>
<tr>
<td>GB-EC (Global block-level [Erasure coding])</td>
<td>6.9</td>
<td>-6.37×</td>
</tr>
</tbody>
</table>
### Deduplication ratio vs. performance

<table>
<thead>
<tr>
<th>Mode</th>
<th>Dedup. ratio</th>
<th>Performance improvement (P-servers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-mode 1</td>
<td>1.5</td>
<td>1.6×</td>
</tr>
<tr>
<td>S-mode</td>
<td>1.3</td>
<td>2×</td>
</tr>
<tr>
<td>B-mode 2</td>
<td>1.2</td>
<td>2.6×</td>
</tr>
<tr>
<td>B-mode 3</td>
<td>1</td>
<td>2.8×</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B-mode 0</th>
<th>Dedup ratio</th>
<th>Performance degradation (D-servers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GF-R (Global file-level [3 replicas])</td>
<td>2.1</td>
<td>-1.03 ×</td>
</tr>
<tr>
<td>GF+LB-R (Global file- and local block-level [3 replicas])</td>
<td>3.0</td>
<td>-2.87 ×</td>
</tr>
<tr>
<td>GB-EC (Global block-level [Erasure coding])</td>
<td>6.9</td>
<td>-6.37 ×</td>
</tr>
</tbody>
</table>
### Deduplication ratio vs. performance

<table>
<thead>
<tr>
<th>Mode</th>
<th>Dedup. ratio</th>
<th>Performance improvement (P-servers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-mode 1</td>
<td>1.5</td>
<td>1.6×</td>
</tr>
<tr>
<td>S-mode</td>
<td>1.3</td>
<td>2×</td>
</tr>
<tr>
<td>B-mode 2</td>
<td>1.2</td>
<td>2.6×</td>
</tr>
<tr>
<td>B-mode 3</td>
<td>1</td>
<td>2.8×</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B-mode 0</th>
<th>Dedup ratio</th>
<th>Performance degradation (D-servers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GF-R (Global file-level [3 replicas])</td>
<td>2.1</td>
<td>-1.03 ×</td>
</tr>
<tr>
<td>GF+LB-R (Global file- and local block-level [3 replicas])</td>
<td>3.0</td>
<td>-2.87×</td>
</tr>
<tr>
<td>GB-EC (Global block-level [Erasure coding])</td>
<td>6.9</td>
<td>-6.37×</td>
</tr>
</tbody>
</table>
## Deduplication ratio vs. performance

<table>
<thead>
<tr>
<th>Mode</th>
<th>Dedup. ratio</th>
<th>Performance improvement (P-servers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-mode 1</td>
<td>1.5</td>
<td>1.6×</td>
</tr>
<tr>
<td>S-mode</td>
<td>1.3</td>
<td>2×</td>
</tr>
<tr>
<td>B-mode 2</td>
<td>1.2</td>
<td>2.6×</td>
</tr>
<tr>
<td>B-mode 3</td>
<td>1</td>
<td>2.8×</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B-mode 0</th>
<th>Dedup ratio</th>
<th>Performance degradation (D-servers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GF-R (Global file-level [3 replicas])</td>
<td>2.1</td>
<td>-1.03 ×</td>
</tr>
<tr>
<td>GF+LB-R (Global file- and local block-level [3 replicas])</td>
<td>3.0</td>
<td>-2.87×</td>
</tr>
<tr>
<td>GB-EC (Global block-level [Erasure coding])</td>
<td>6.9</td>
<td>-6.37×</td>
</tr>
</tbody>
</table>
Deduplication ratio vs. performance

<table>
<thead>
<tr>
<th>Mode</th>
<th>Dedup. ratio</th>
<th>Performance improvement (P-servers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-mode 1</td>
<td>1.5</td>
<td>1.6×</td>
</tr>
<tr>
<td>S-mode</td>
<td>1.3</td>
<td>2×</td>
</tr>
<tr>
<td>B-mode 2</td>
<td>1.2</td>
<td>2.6×</td>
</tr>
<tr>
<td>B-mode 3</td>
<td>1</td>
<td>2.8×</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B-mode 0</th>
<th>Dedup ratio</th>
<th>Performance degradation (D-servers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GF-R (Global file-level [3 replicas])</td>
<td>2.1</td>
<td>-1.03 ×</td>
</tr>
<tr>
<td>GF+LB-R (Global file- and local block-level [3 replicas])</td>
<td>3.0</td>
<td>-2.87×</td>
</tr>
<tr>
<td>GB-EC (Global block-level [Erasure coding])</td>
<td>6.9</td>
<td>-6.37×</td>
</tr>
</tbody>
</table>
Prefetch cache hit ratio

State of the art

DupHunter
Prefetch cache hit ratio

DupHunter can provide high hit ratio while reducing tail latency.
Preconstruct cache hit ratio

% of GET layer requests

<table>
<thead>
<tr>
<th></th>
<th>Dal</th>
<th>Fra</th>
<th>Lon</th>
<th>Syd</th>
</tr>
</thead>
<tbody>
<tr>
<td>GF-R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GF+LB-R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GB-EC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend:
- **Hit**
- **Wait**
- **Miss**
Preconstruct cache hit ratio

Global file level deduplication also has the lowest wait and miss ratios.

Preconstruct cache hit ratio

% of GET layer requests

Hit  Wait  Miss


Global file level deduplication also has the lowest wait and miss ratios.

Nannan Zhao  znannan1@vt.edu
Summary

- DupHunter exploits the redundancy in container images along with predictable user access patterns to achieve high space savings with low layer restore overhead.
  - It supports multiple replica deduplication modes.
  - It facilitates parallel layer reconstruction.
  - It offers proactive layer prefetching/preconstruction.

- DupHunter reduces storage space needs by up to 6.9x and can reduce the GET layer latency up to 2.8x compared to the state of the art.

- DupHunter is available at [https://github.com/nnzhaocs/DupHunter](https://github.com/nnzhaocs/DupHunter).
THANK YOU

Questions: Nannan Zhao znannan1@vt.edu

DSSL@VT: http://dssl.cs.vt.edu